Schema Theory for Genetic Programming with One-point Crossover and Point Mutation

Riccardo Poli and William B. Langdon

School of Computer Science

The University of Birmingham

Birmingham B15 2TT, UK

E-mail: {R.Poli,W.B.Langdon}@cs.bham.ac.uk

Telephone: +44-121-414-{3739,4791}

FAX: +44-121-414-4281

Keywords: genetic programming, schema theorem, one-point crossover

Abstract

We review the main results obtained in the theory of schemata in Genetic Programming (GP) emphasising their strengths and weaknesses. Then we propose a new, simpler definition of the concept of schema for GP which is closer to the original concept of schema in genetic algorithms (GAs). Along with a new form of crossover, one-point crossover, and point mutation this concept of schema has been used to derive an improved schema theorem for GP which describes the propagation of schemata from one generation to the next. We discuss this result and show that our schema theorem is the natural counterpart for GP of the schema theorem for GAs, to which it asymptotically converges.

1 Introduction

Genetic Programming (GP) has been applied successfully to a large number of difficult problems like automatic design, pattern recognition, robotic control, synthesis on neural architectures, symbolic regression, music and picture generation [2, 9, 10, 11, 12, 13]. However a relatively small number of theoretical results are available to try and explain why it works and how (see [14, pages 517–519] for a list of references).

Since John Holland's seminal work in the mid seventies and his well known schema theorem (see [6] and [5]), schemata are often used to explain why GAs work. Schemata are similarity templates representing entire groups of chromosomes. The schema theorem describes how schemata are expected to propagate generation after generation under the effects of selection, crossover and mutation. One of the reasons why schemata are considered important is that, under certain conditions, the schema theorem can be used to prove that GAs realise an optimum search strategy. The usefulness of schemata and the optimality of GA search have been recently criticised (see for example [1, 15, 7, 8]). However, the schema theorem is still a widely accepted description of the way GAs search. So, a natural way of creating a theory for GP is to define a concept of schema for parse trees and to extend Holland's schema theorem to GP.

One of the difficulties in obtaining theoretical results using the idea of schema is that the definition of schema for GP is much less straightforward than for GAs and a few alternative definitions have been proposed in the literature. All of them define schemata as composed of one or multiple trees or fragments of trees. In some definitions [10, 18, 19, 28, 29] schema components are *non-rooted* and, therefore, a schema can be present multiple times within the same program. This, together with the variability of the size and shape of the programs matching the same schema, leads to some complications in the computation of the schema-disruption probabilities necessary to formulate schema theorems for GP. In a recent definition [25] schemata are represented by *rooted* tree fragments. This definition makes schema theorem calculations easier. We will critically review the main results obtained to date in the theory of schemata for GP in Section 2 where we will also briefly recall Holland's schema theory for binary GAs.

In Section 3 we propose a new simpler definition of schema for GP where a schema is represented by a single rooted tree. This concept of schema, which is much closer to the original concept of schema in GAs, suggested a simpler form of crossover for GP in which *the same* crossover point is selected in both parents. We call this new operator one-point crossover because of its similarity with the corresponding GA operator. This is described in Section 4. In Section 5, we have used these tools to derive a simple and natural schema theorem, in which the probability of disruption for a schema can be estimated very easily. An analysis of the theoretical results described in the paper is reported in Section 6 while Section 7 includes some final remarks and indications of future work.

2 Previous Work on Schemata for GP

As highlighted in [23], a schema is a subspace of the search space. Usually schemata are written in some language using a concise notation rather than as ordinary sets, which would require listing all the points of the search space they contain: an infeasible task even for relatively small search spaces.

2.1 GA schemata

In the context of GAs operating on binary strings, a schema (or similarity template) is a string of symbols taken from the alphabet {0,1,#}. The character # is interpreted as a "don't care" symbol, so that a schema can represent several bit strings. For example the schema #10#1 represents four strings: 01001, 01011, 11001 and 11011. The number of non-# symbols is called the *order* $\mathcal{O}(H)$ of a schema H. The distance between the furthest two non-# symbols is called the *defining length* $\mathcal{L}(H)$ of the schema. Holland obtained a result (the schema theorem) which predicts how the number of strings in a population matching (or belonging to) a schema is expected to vary from one generation to the next [6]. The theorem is as follows:

$$E[m(H,t+1)] \geq m(H,t) \cdot \underbrace{\frac{f(H,t)}{\bar{f}(t)}}_{Selection} \cdot \underbrace{(1-p_m)^{\mathcal{O}(H)}}_{Mutation} \cdot \underbrace{\left[1 - p_c \overbrace{\frac{\mathcal{L}(H)}{N-1}\left(1 - \frac{m(H,t)f(H,t)}{M\bar{f}(t)}\right)}^{P_d(H,t)}\right]}_{Crossover} \qquad (1)$$

where $m(H,t)$ is the number of strings matching the schema H at generation t, $f(H,t)$ is the mean fitness of the strings matching H, $\bar{f}(t)$ is the mean fitness of the strings in the population, p_m is the probability of mutation per bit, p_c is the probability of crossover, N is the number of bits in the strings, M is the number of strings in the population, and $E[m(H,t+1)]$ is the expected number of strings matching the schema H at generation $t+1$.[1] The three horizontal curly brackets beneath the equation indicate which operators are responsible for each term. The bracket above the equation represents the probability of disruption of the schema H at generation t due to crossover $P_d(H,t)$. Such a probability depends on the frequency of the schema in the mating pool but also on the intrinsic *fragility* of the schema $\frac{\mathcal{L}(H)}{N-1}$.

[1]This is a slightly different version of Holland's original theorem. Equation 1 applies when crossover is performed taking both parents from the mating pool [5, 31].

Search algorithms exploring an unknown space will have to compromise between exploration and exploitation. Exploration is needed to gain knowledge about the search space to decide which areas are more promising. Exploitation represents the desire to obtain good solutions as quickly as possible (for example because they are actually being used while the search goes on). Assuming the ratio between the fitness of a schema and the population fitness does not change, Holland's schema theorem predicts that the genetic material contained in chromosomes belonging to good schemata will spread in the population at an exponential rate. In other words $m(H,t)$ will grow exponentially with the number of generations t [6, 5]. Under certain hypotheses this strategy represents the optimum compromise between the need for exploration and the need for exploitation. This can be shown using the analogy between GAs and multi-armed bandits according to which the operations performed by a GA on schemata are equivalent to playing many multi-armed bandits [6, 5, 3]. Unfortunately, the assumptions required to prove the optimality of GA search in this way are difficult to maintain particularly because of the presence of cross-competition between schemata and the fact that the runtime estimates of a schema's fitness may not represent the true quality of the corresponding subspace [17, 15, 7, 8, 27, 32].

2.2 From GA schemata to GP schemata

In this section we discuss and modify the representation of schemata used in GAs to make it easier for the reader to understand some of the differences between the GP schema definitions introduced in the following sections.

A GA schema is fully determined by the defining bits (the 0's and 1's) it contains and by their position. So, instead of representing a schema as a string of characters, one could equivalently represent it with a set of pairs $\{(c_1, i_1), (c_2, i_2), \ldots\}$. The terms c_j are strings of 0's and 1's which represent groups of contiguous defining bits which we call the *components* of the schema. The terms i_j are integers giving the positions of the c_j's. For example the schema #10#1 would have two components and could be represented as $\{(\texttt{10},2),(\texttt{1},5)\}$.

This is an explicit representation for a way of looking at schemata as sets of components that has been implicitly used often in the GA literature. For example, when we say that a schema has been disrupted by crossover, we usually mean that one or more of its components have not been transmitted to the offspring entirely, or have been partly destroyed, rather than thinking that the offspring sample a subspace different from the one represented by the schema. Likewise, we explain the *building block hypothesis* [5, 6] by saying that GAs work by hierarchically composing relatively fit, short schemata to form complete solutions. What we mean is that crossover mixes and concatenates the components of low order schemata to form higher order ones, we do not mean that crossover is allocating more samples to higher-order schemata representing the intersection of good lower-order ones. So, the GA schema theorem is often interpreted as describing the variations of the number of instances of certain groups of bits within the population (the schema components c_j, at their positions i_j) rather than the number of strings sampling the subspace represented by a schema. For example, if $H = \#10\#1$ the theorem could be interpreted as describing the variations of frequency of the component 10 at position 2 and 1 at position 5 within a population of bit strings, rather than the variations in the proportion of individuals belonging to the set H.

Obviously there is no real distinction between these two ways of interpreting the schema theorem, because counting the number of strings matching (or belonging to) a given schema and counting the number of schema components in the population give the same result. This is because we specify exactly where each component of the schema must be located.

Now, let us try to understand what would happen if we omitted the positional information i_j from all schema pairs. Syntactically schemata would be sets of components $\{c_1, c_2, \ldots\}$, like H={10,1}. Semantically a schema would represent all the strings which include as substrings all c_j's. For example the schema H= {10,1} would represent 11000, 10100, 11110, etc. Interestingly, with this definition of schema a string can include multiple copies of the components of a schema. For example, the component c_1=10 of the previous schema is present two times (at positions 1 and 3) in the string 10101. Similarly the component c_2=1 is present three times. So, c_1 and c_2 are jointly present in the string in six different ways. We call each of these ways an *instantiation* of the schema in the string. Therefore, if the positional information is omitted from a GA schema definition, counting the number of strings belonging to a given schema and counting the number of instantiations of the schema produce different results. This means that the two ways of interpreting the schema theorem mentioned in the previous paragraphs are different.

The previous argument might look academic for binary fixed-length GAs and most people would think that position-less schema definitions are not very useful for GAs and GP in general. However, position-less schemata might be useful when one uses more complex representations and gene-position-independent fitness functions. For example, a position-less schema-component representation similar to the one presented above has been proposed by Radcliffe [23] to represent sets of solutions of travelling sales-person problems (edge formae).

The distinction between positioned and position-less schema components is also very important in GP.

In fact information about the positions of the schema components was omitted in the earlier GP schema definitions reviewed below. This led some authors to concentrate their analysis on the propagation of such components in the population rather than on the way the number of programs sampling a given schema change over time.

2.3 Koza's GP schemata

Given the popularity of Holland's work, Koza [10, pages 116–119] made a first attempt to explain why GP works producing an informal argument showing that Holland's schema theorem would work for GP as well. The argument was based on the idea of defining a schema as the subspace of all trees which contain a predefined set of subtrees. According to Koza's definition a schema H is represented as a set of S-expressions. For example the schema H={`(+ 1 x), (* x y)`} represents all programs including at least one occurrence of the expression `(+ 1 x)` and at least one occurrence of `(* x y)`. This definition of schema was probably suggested by the fact that Koza's GP crossover moves around subtrees. Koza's definition gives only the defining components of a schema not their position, so the same schema can be instantiated (matched) in different ways, and therefore multiple times, in the same program. For example, the schema H={`x`} can be instantiated in two ways in the program `(+ x x)`. (One of the two possible instantiations of the more complex schema H={`(- 2 x), x`} in the program `(+ (- 2 x) x)` is shown in Figure 1(a).)

2.4 O'Reilly's GP schemata

Koza's work on schemata was later formalised and refined by O'Reilly [18, 19] who derived a schema theorem for GP in the presence of fitness-proportionate selection and crossover. The theorem was based on the idea of defining a schema as an unordered collection (a multiset) of subtrees and tree fragments. Tree fragments are trees with at least one leaf that is a "don't care" symbol ('#') which can be matched by any subtree (including subtrees with only one node). For example the schema H={`(+ # x), (* x y), (* x y)`} represents all the programs including at least one occurrence of the tree fragment `(+ # x)` and at least *two* occurrences of `(* x y)`.[2] The tree fragment `(+ # x)` is present in all programs which include a `+` the second argument of which is `x`. Figure 1(b) shows the instantiation of the schema H={`(+ # x), x, x`} in the program `(+ (- 2 x) x)`. Like Koza's definition O'Reilly's schema definition gives only the defining components of a schema not their position. So, again the same schema can be instantiated in different ways, and therefore

[2] We use here the standard notation for multisets, which is slightly different from the one used in O'Reilly's work.

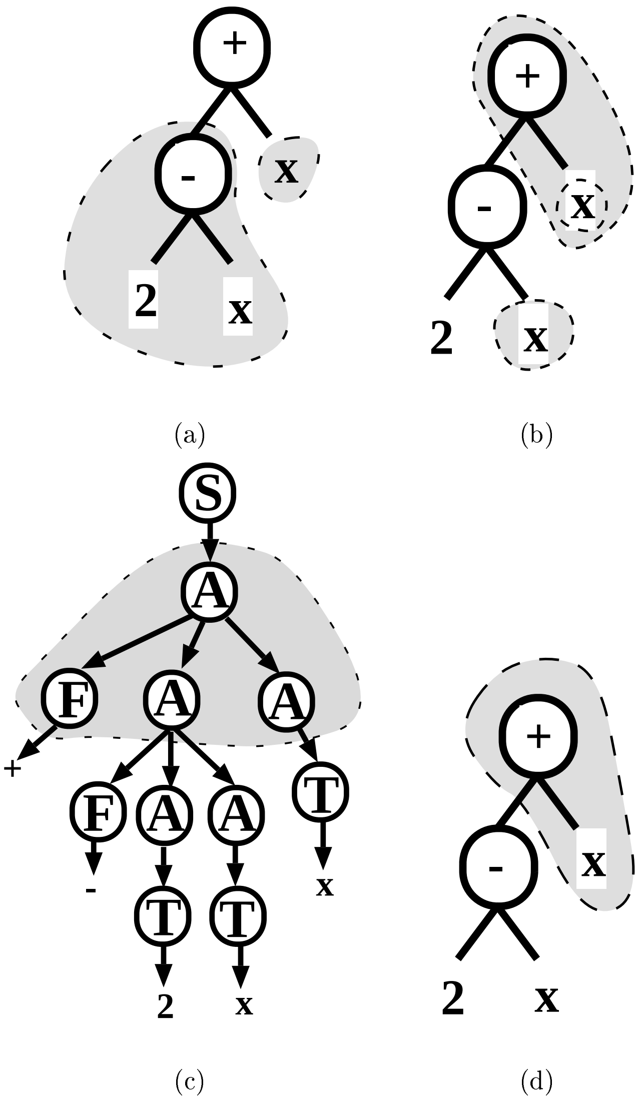


Figure 1: Instantiations of: (a) Koza's schema H={`(- 2 x), x`}, (b) O'Reilly's schema H={`(+ # x), x, x` }, (c) Whigham's schema $H = (\mathtt{A} \overset{+}{\Rightarrow} \mathtt{FAA})$ and (d) Rosca's schema H=`(+ # x)`. The shaded regions represent the component(s) of the schema.

multiple times, in the same program.

O'Reilly's definition of schema allowed her to define the concept of order and defining length for GP schemata. In her definition the order of a schema is the number of non-# nodes in the expressions or fragments contained in the schema. The defining length is the number of links included in the expressions and tree fragments in the schema plus the links which connect them together. Unfortunately, the definition of defining length is complicated by the fact that the components of a schema can be embedded in different ways in different programs. Therefore, the defining length of a schema is not constant but depends on the way a schema is instantiated inside the programs sampling it. The probability of disruption $P_d(H,h,t)$ of a schema H contained in the program h at generation t due to crossover is the ratio between the defining length of H in h and the total number of crossover locations in h. This implies that $P_d(H,h,t)$ depends on the shape, size and composition of the tree h matching the schema. The schema theorem derived by O'Reilly

$$E[i(H,t+1)] \geq i(H,t) \cdot \frac{f(H,t)}{\bar{f}(t)} \cdot \left(1 - p_c \cdot \overbrace{\max_{h \in \mathrm{Pop(t)}} P_d(H,h,t)}^{P_d(H,t)}\right) \tag{2}$$

overcame this problem by considering the maximum of such a probability, $P_d(H,t) = max_{h \in \mathrm{Pop(t)}} P_d(H,h,t)$ which may lead to severely underestimating the number of occurrences of the given schema in the next generation. It is important to note $i(H,t)$ is the number of *instances* of the schema H at generation t and $f(H,t)$ is the mean fitness of the instances of H. This is computed as the weighted sum of the fitnesses of the programs matching H, using as weights the ratios between the number of instances of H each program contains and the total number of instances of H in the population. The theorem describes the way in which the components of the representation of a schema propagate from one generation to the next, rather than the way the number of programs sampling a given schema change during time. O'Reilly discussed the usefulness of her result and argued that the intrinsic variability of $P_d(H,t)$ from generation to generation is one of the major reasons why no hypothesis can be made on the real propagation and use of building blocks (short, low-order relatively fit schemata) in GP. O'Reilly's schema theorem did not include the effects of mutation.

2.5 Whigham's GP schemata

In the framework of his GP system based on context free grammars (CFG-GP) Whigham produced a concept of schema for context-free grammars and the related schema theorem [28, 30, 29]. In CFG-GP programs are the result of applying a set of rewrite rules taken from a pre-defined grammar to a starting symbol S. The process of creation of a program can be represented with a derivation tree whose internal nodes are rewrite rules and whose terminals are the functions and terminals used in the program. In CFG-GP the individuals in the population are derivation trees and the search proceeds using crossover and mutation operators specialised so as to always produce valid derivation trees.

Whigham defines a schema as a partial derivation tree rooted in some non-terminal node, i.e. as a collection of rewrite rules organised into a single derivation tree. Given that the terminals of a schema can be both terminal and non-terminal symbols of a grammar and that the root of a schema can be a symbol different from the starting symbol S, a schema represents all the programs that can be obtained by completing the schema (i.e. by adding other rules to its leaves until only terminal symbols are present) and all the programs represented by schemata which contain it as a component. When the root node of a schema is not S, the schema can occur multiple times in the derivation tree of the same program. This is the result of the absence of positional information in the schema definition. Figure 1(c) shows one of the two possible instantiations of the schema $H = (\mathtt{A} \overset{+}{\Rightarrow} \mathtt{FAA})$ in the derivation tree of the program `(+ (- 2 x) x)`.

Whigham's definition of schema leads to simple equations for the probability of disruption of schemata under crossover, $P_{d_c}(H,h,t)$, and mutation, $P_{d_m}(H,h,t)$. Unfortunately as with O'Reilly's, these probabilities vary with the size of the tree h matching the schema. In order to produce a schema theorem for CFG-GP Whigham used the average disruption probabilities of the instances of a schema under crossover and mutation, $\bar{P}_{d_c}(H,t)$ and $\bar{P}_{d_m}(H,t)$, and the average fitness $f(H,t)$ of such instances. The theorem is as follows:

$$E[i(H,t+1)] \geq i(H,t)\frac{f(H,t)}{\bar{f}(t)} \cdot \left\{[1 - p_m\bar{P}_{d_m}(H,t)]\,[1 - p_c\bar{P}_{d_c}(H,t)]\right\}, \tag{3}$$

where p_c and p_m are the probabilities of applying crossover and mutation. By changing the grammar used in CFG-GP this theorem can be shown to be applicable both to GAs with fixed length binary strings and to standard GP, of which CFG-GP is a generalisation (see the GP grammar given in [29, page 130]). Like in O'Reilly's case, this theorem describes the way in which the components of the representation of a schema propagate from one generation to the next, rather than the way the number of programs sampling a given schema change over time. The GP schema theorem obtained by Whigham is different from the one obtained by O'Reilly as the concept of schema used by the two authors is different. Whigham's schemata represent derivation-tree fragments which always represent single subexpressions, while O'Reilly's schemata can represent multiple subexpressions.

2.6 Rosca's GP schemata

More recently Rosca [25] has proposed a new definition of schema, called *rooted tree-schema*, in which a schema is a rooted contiguous tree fragment. For example, the rooted tree-schema H=`(+ # x)` represents all the programs whose root node is a `+` the second argument of which is `x`. Figure 1(d) shows the instantiation of this schema in the program `(+ (- 2 x) x)`. The rootedness of this schema representation, which was developed at the same time as and independently from ours, is very important as it reintroduces in the schema definition the positional information lacking in the previous definitions of schema for GP. As a consequence a schema can be instantiated at most once within a program and studying the propagation of the components of the schema in the population is equivalent to analysing the way the number of programs sampling the schema change over time. Rosca derived the following schema theorem:

$$E[m(H,t+1)] \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)} \cdot \left[1 - (p_m + p_c) \underbrace{\sum_{h \in H \cap Pop(t)} \frac{\overbrace{\mathcal{O}(H)}^{P_d(H,h,t)}}{N(h)} \frac{f(h)}{\sum_{h \in H \cap Pop(t)} f(h)}}_{P_d(H,t)} \right], \qquad (4)$$

where $N(h)$ is the size of a program h matching the schema H, $f(h)$ is its fitness, and the order of a schema $\mathcal{O}(H)$ is the number of defining symbols it contains.[3] Rosca did not give a definition of defining length for a schema. Rosca's schema theorem involves the evaluation of the weighted sum of the fragilities $\frac{\mathcal{O}(H)}{N(h)}$ of the instances of a schema within the population, using as weights the ratios between the fitness of the instances of H and the sum of the fitness of such instances.

In the three definitions of GP schema mentioned above, in general schemata divide the space of programs into subspaces containing programs of different sizes and shapes. In the next section we give a new definition of schema which partitions the program space into subspaces of programs of fixed size and shape.

3 Definition of Schema for GP

If we take the viewpoint in which a schema is a subspace of the space of possible solutions then we can see schemata as mathematical tools to describe which areas of the search space are sampled by a population. For schemata to be useful in explaining how GP searches, their definition must make the effects of selection, crossover and mutation comprehensible and relatively easy calculate. The problem with some of the earlier

[3]We have rewritten the theorem in a form which is slightly different from the original one to highlight some of its features.

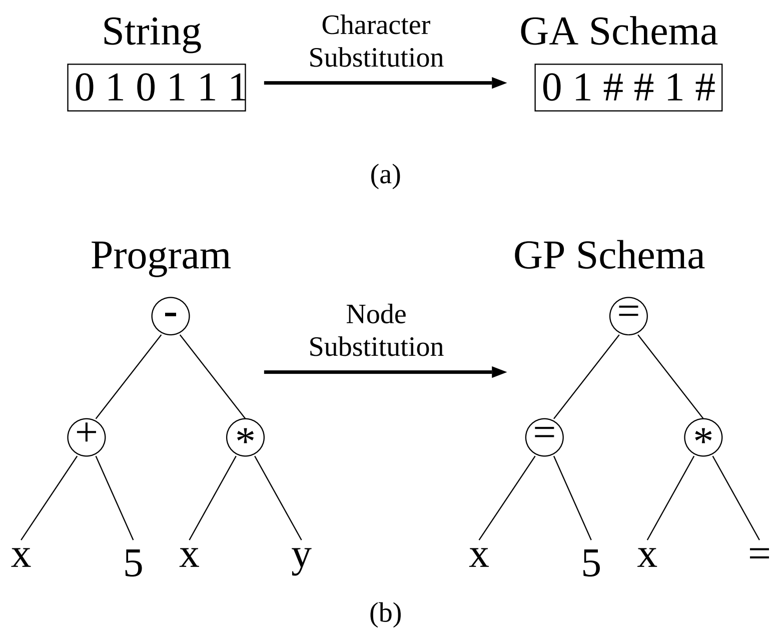


Figure 2: Relation between (a) bit strings and the GA schemata they sample and (b) programs and the GP schemata they sample.

definitions of schemata for GP is that they make the effects on schemata of the genetic operators used in GP too difficult to evaluate. In this section we will describe a new definition of schema for GP which allowed us to overcome this problem.

Our definition of GP schema was inspired by the definition of schema for linear GAs. In standard fixed length linear GAs it is possible to obtain all the schemata contained in a string by simply replacing, in all possible ways, its symbols with "don't care" symbols which represent a *single character*. This process is shown in Figure 2(a) for a 6-bit string. Intuitively, exactly the same approach can be used to obtain the GP schemata sampled by a program. It is sufficient to replace the nodes in a program with "don't care" nodes representing exactly *a single function or terminal* as in Figure 2(b). In the figure the equal sign "=" is used to as a "don't care" symbol to emphasise the difference between our definition of schema and other GP definitions where the "don't care" symbol # is used to represent entire subtrees.

A natural extension to GP of the GA schema definition is:

Definition 1 (GP schema). *A* GP schema *is a rooted tree composed of nodes from the set $\mathcal{F} \cup \mathcal{T} \cup \{=\}$, where $\mathcal{F}$ and $\mathcal{T}$ are the function set and the terminal set used in a GP run, and the operator* = *is a polymorphic function with as many arities as the number of different arities of the elements of $\mathcal{F} \cup \mathcal{T}$, the terminals in*

$\mathcal{T}$ being 0-arity functions.

In this definition we could have equivalently used different symbols for "don't care" symbols with different arities instead of a single polymorphic "don't care" symbol. However, given that the arity of each `=` sign in a schema can be inferred by counting its arguments, for the sake of simplicity we have decided not to do so. The specification of the properties of the "don't care" symbol given in Definition 1 is not a complication due to the variable size nature of the representation used in GP: it would be necessary also for linear GAs if different loci could take a different number of alleles (i.e. in a multi-cardinality GA). There may be cases in which a "don't care" symbol in one of our GP schemata can only stand for a particular function or terminal. This could happen if there is only one function in $\mathcal{F} \cup \mathcal{T}$ with a particular arity. The same thing would happen in a multi-cardinality GA if a particular locus could only take one allele. The theory presented in the paper could readily be extended to cover this special case.

In line with the original definition of schema for GAs, a GP schema H represents multiple programs, all having the same shape as the tree representing H and the same labels for the non-= nodes. For example, if $\mathcal{F}$={`+, -`} and $\mathcal{T}$={`x, y`} the schema H=`(+ (- = y) =)` would represent the four programs `(+ (- x y) x)`, `(+ (- x y) y)`, `(+ (- y y) x)` and `(+ (- y y) y)`.

We can now extend the concepts of order, length and defining length to our GP schemata.

Definition 2 (Order). *The number of non-= symbols is called the* order $\mathcal{O}(H)$ *of a schema H.*

Definition 3 (Length). *The total number of nodes in the schema is called the* length $N(H)$ *of a schema H.*[4]

Definition 4 (Defining Length). *The number of links in the minimum tree fragment including all the non-= symbols within a schema H is called the* defining length $\mathcal{L}(H)$ *of the schema.*

Like in traditional binary GAs our definitions of order, length and defining length of a schema are totally independent on the shape and size of the programs in the actual population.

Schemata of low order and large length can represent a huge number of programs. For example, if $\mathcal{F}$ includes only two functions and $\mathcal{T}$ only two terminals, a schema of order $\mathcal{O} = 5$ and length $N = 30$ represents $2^{N-\mathcal{O}} = 33,554,432$ different programs. Other definitions of schema for GP have this property with the difference that the number of programs represented by our schemata is *finite* and can easily be computed (if the function set and the terminal set are finite), while such a number is infinite with other schema definitions

[4]The length of a schema H is also equal to the size (number of nodes) of the programs matching H.

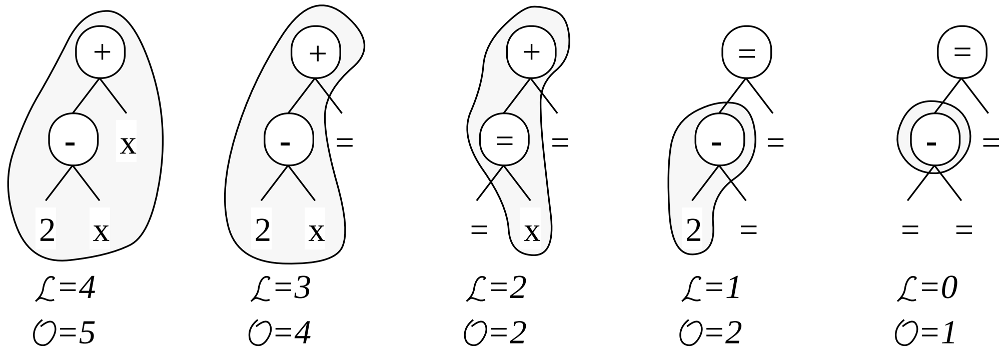


Figure 3: Some schemata sampled by the program `(+ (- 2 x) x)`.

(unless the depth or size of the programs is limited, in which case the number is finite but cannot be expressed in a simple closed form).

Likewise, given that a program of length N includes 2^N different schemata, a modest-size population of programs can actually sample a huge number of schemata. Computing a lower and an upper bound of such a number is trivial with our definition (there are between $2^{N_{\min}}$ and $M \cdot 2^{N_{\max}}$ schemata in a population of size M whose smallest and largest trees include N_{min} and N_{max} nodes, respectively), while this is quite difficult with other definitions of schema.

The defining length $\mathcal{L}(H)$ can be computed with a simple recursive procedure, not dissimilar from the one necessary to compute $N(H)$. Figure 3 shows some of the 32 schemata matching the program `(+ (- 2 x) x)` along with their order $\mathcal{O}$ and defining length $\mathcal{L}$. The minimum tree fragments used to compute $\mathcal{L}$ are enclosed in the shaded regions.

Our definition of schema is in some sense lower-level than those adopted by O'Reilly, Whigham and Rosca as a smaller number of trees can be represented by schemata with the same number of "don't care" symbols and it is possible to represent other kinds of schemata by using a collection of ours. For example, assuming that the maximum allowed depth for trees is 2 and that all functions have arity 2, O'Reilly's schema H={`(+ # #)`} can be represented by our schemata `(+ = =)`, `(= = (+ = =))`, `(= (+ = =) =)`, `(= (+ = =) (= = =))`, `(= (= = =) (+ = =))`, `(+ = (= = =))`, `(+ (= = =) =)` and `(+ (= = =) (= = =))`. The converse is not true, as, for example, the similarity template `(= x y)` cannot be represented using O'Reilly's schemata. Similarly, it is easy to show that Rosca's schemata can be represented by sets of our schemata. The converse is not true as all the schemata whose defining nodes do not form a compact subtree (i.e. for which $\mathcal{L}(H) > \mathcal{O}(H) - 1$) cannot be represented using Rosca's schemata.

Our definition of schema, like Rosca's, reintroduces the positional information lacking in previous definitions of schema for GP. As a consequence the number of instances of a schema in the population coincides with the number of programs sampling the schema.

In order to make it easier to understand the effects on schemata of one-point crossover it is useful to introduce two more definitions which allow us to distinguish two broad classes of schemata.

Definition 5 (Hyperspaces and Hyperplanes). *A schema G is a* hyperspace *if it does not contain any defining node (i.e. if $\mathcal{O}(G) = 0$). A schema H is a* hyperplane *if it contains at least one defining node. The schema $G(H)$ obtained by replacing all the defining nodes in a hyperplane H with "don't care" symbols is called the* hyperspace associated to H.

Figure 4 represents the relation between programs, hyperplanes (ordinary schemata) and hyperspaces of programs. Each hyperspace represents all the programs with a given shape. Hyperspaces form a tessellation of the space of possible programs, i.e. they subdivide it into non-overlapping subsets whose union is the original space. Unless a depth or size limit is imposed on the programs being considered, there are infinitely many hyperspaces. However, the number of programs belonging to each hyperspace is finite (unless $\mathcal{F}$ or $\mathcal{T}$ are infinite). Hyperplanes are smaller subsets of programs. Their name derives from the geometric interpretation illustrated in Figure 5 in which schemata of order 0 (hyperspaces) are represented as multidimensional hyper-parallelepipeds the edges and faces of which represent schemata with order greater than 0 (i.e. hyperplanes). This interpretation is a direct extension of the usual interpretation of GA schemata as faces and edges of the unit hypercube the vertices of which are all the possible bit strings of a prefixed length.

4 Point Mutation and One-Point Crossover in GP

The concept of schema just introduced is very simple and natural. However, like most previous schema definitions it does not make the effects of standard GP crossover easy to calculate. Given the similarity of our definition with the one of GA schema we started wondering whether it would have been possible to get again inspiration from Holland's work and define more natural crossover and mutation operators.

The obvious analogue of bit-flip mutation is the substitution of a function in the tree with another function with the same arity, or the substitution of a terminal with another terminal: a technique that has been sometimes used in the GP literature [16]. We will call this operation *point mutation*. The perhaps less obvious equivalent of one-point crossover for bit strings (in which a common crossover point is selected in

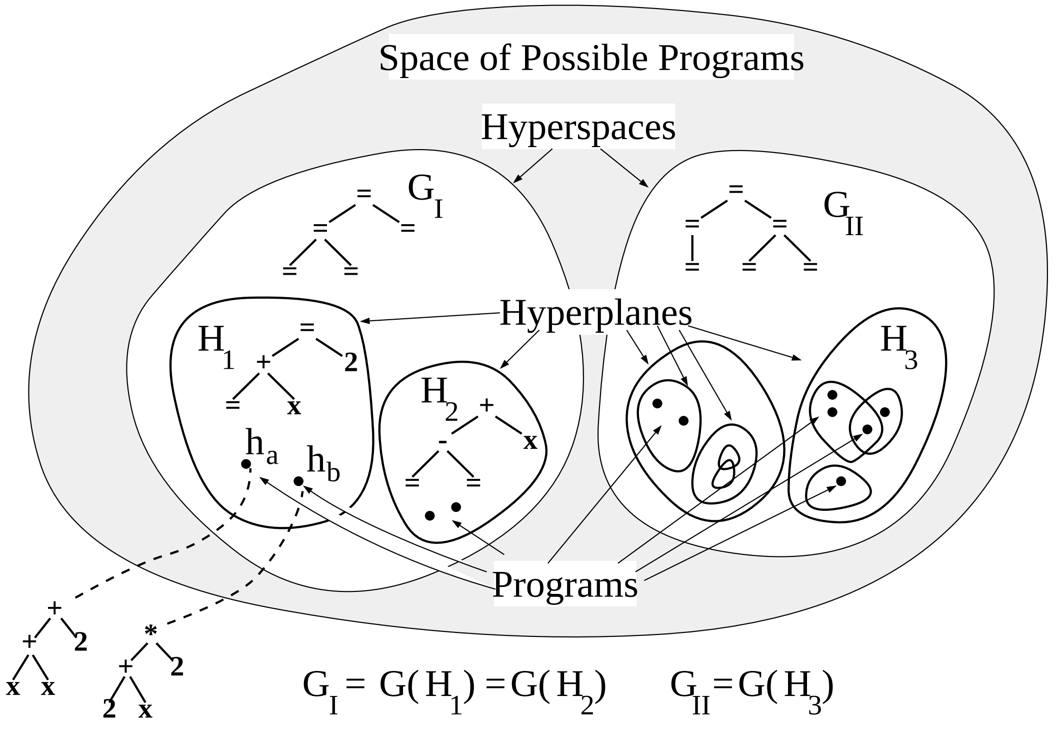


Figure 4: Interpretation of hyperspaces, hyperplanes and programs as sets.

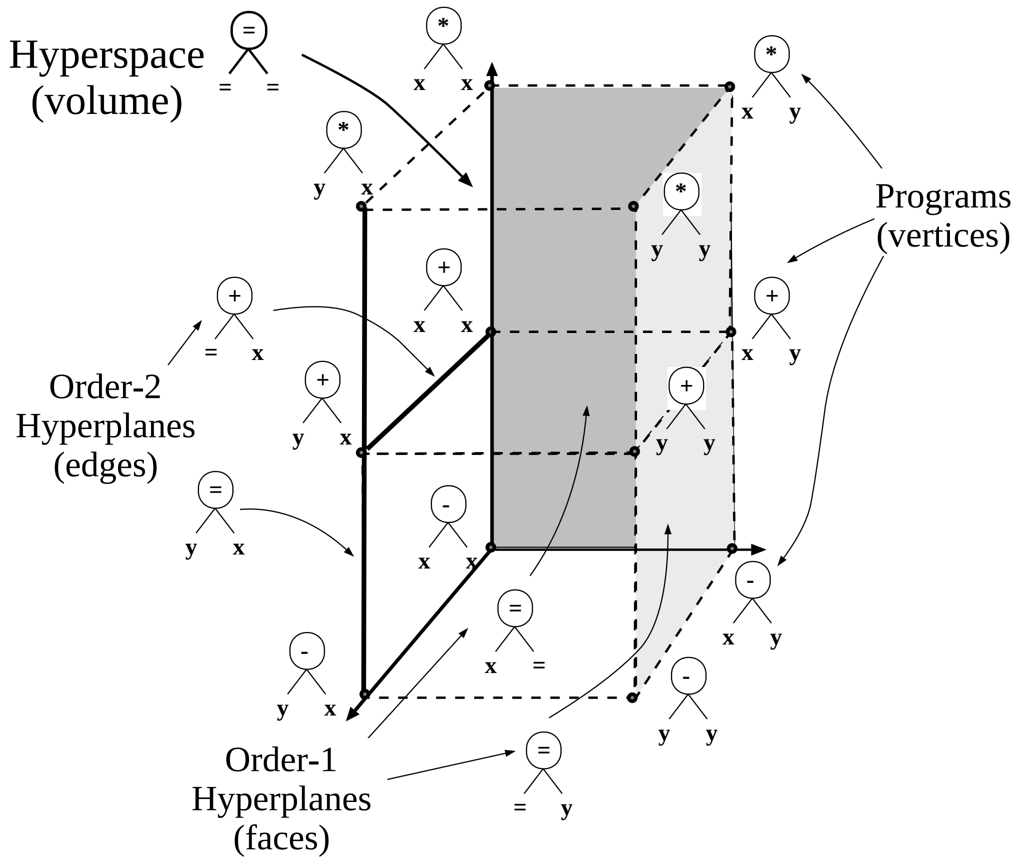


Figure 5: Geometric interpretation of the hyperspace (= = =) when $\mathcal{F}$={-, +, *} and $\mathcal{T}$={x, y}.

both parents, and the offspring are produced by swapping the bits on the right of the crossover point) is a new crossover operator that we also call *one-point crossover*.

The way one-point crossover works was suggested by considering the three steps involved in one-point crossover in binary GAs. These steps are schematised in Figure 6(a). They are: 1) alignment of the parent strings, 2) selection of a common crossover point, 3) swap of the right-end sides of the strings to obtain two offspring. If all the trees in a GP population had exactly the same shape and size then a crossover operator involving exactly the same three steps could easily be implemented. The nodes in the trees would play the role of the digits in the bit strings, the links would play the role of the gaps between digits. Figure 6(b) depicts this situation. In the figure the alignment process corresponds to translating the graphical representation of one of the parents until its root node coincides with the root node of the other.

What would happen if one tried the same procedure with trees having different shapes? To a certain extent it would still be possible to align the trees. After the alignment it would be easy to identify the links which overlap, to select a common crossover point and swap the corresponding subtrees. Our one-point crossover operator for GP is based on these simple steps. They are sketched in Figure 6(c).

From the implementation point of view, the three phases involved in one-point crossover are as follows:

Alignment Copies of the two parent trees are recursively (jointly) traversed starting from the root nodes to identify the parts with the same shape, i.e. with the same arity in the nodes visited. Recursion is stopped as soon as an arity mismatch between corresponding nodes in the two trees is present. All the links encountered are stored.

Crossover Point Selection A random crossover point is selected with a uniform probability among the stored links.

Swap The two subtrees below the common crossover point are swapped in exactly the same way as in standard crossover.

One-point crossover has several interesting features that make it different from standard crossover.

Firstly, in the absence of mutation one-point crossover makes the population converge (like crossover in binary GAs), possibly with help from genetic drift. This can be understood by considering the following. Until a large-enough proportion of the population has exactly the same structure in the upper parts of the tree, the probability of selecting a crossover point in the lower parts is relatively small. This effectively means that until a common upper structure emerges, one-point crossover is actually searching a much smaller space

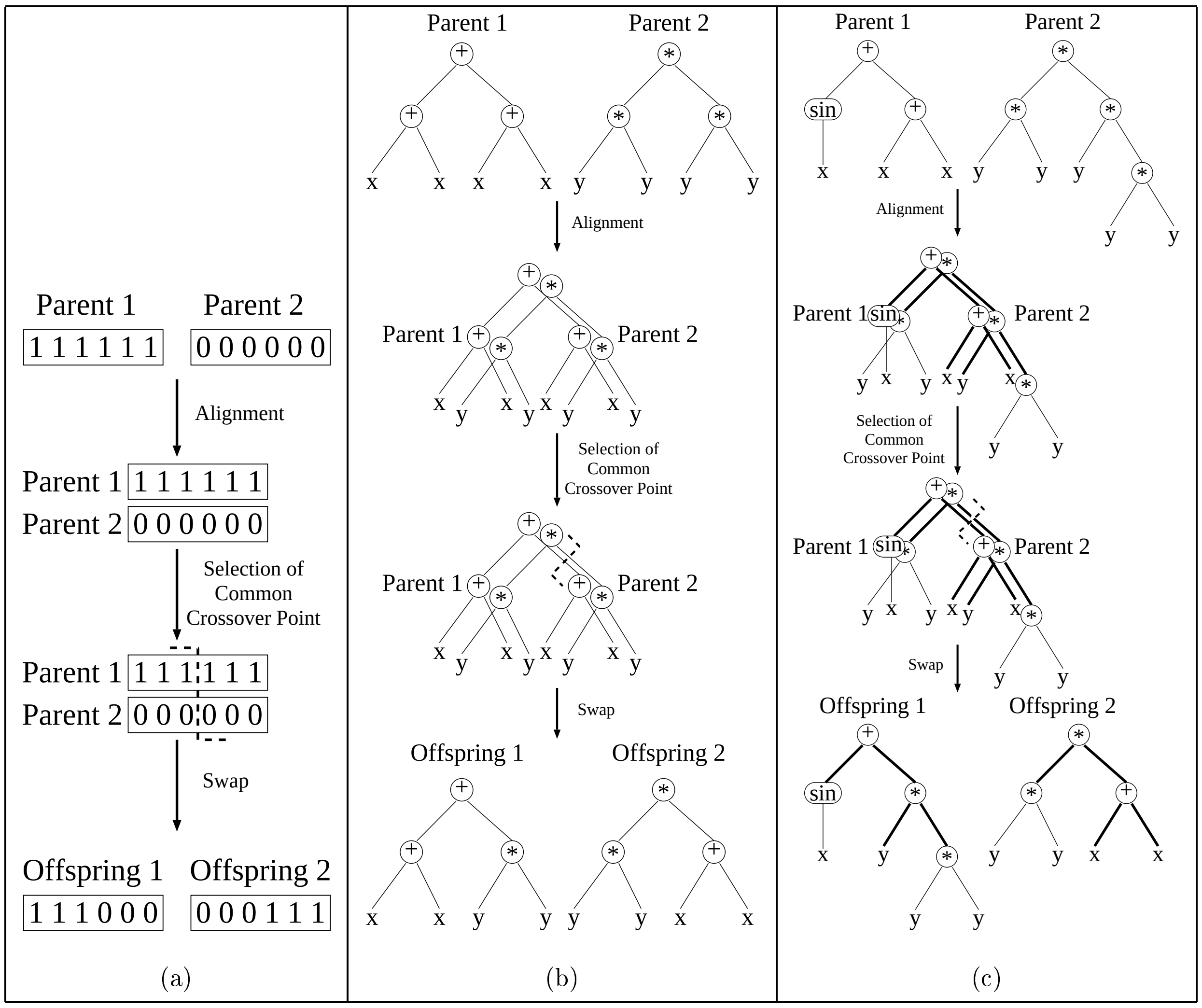


Figure 6: (a) One-point crossover in binary GAs, (b) one-point crossover for GP when both parents have the same shape, and (c) general form of one-point crossover for GP. In (c) the links that can be selected as common crossover points are drawn with thick lines.

of approximately fixed-size (upper) structures. Obviously the lower parts of the subtrees moved around by crossover influence the fitness of the fixed-size upper parts, but they are not modified by crossover at this stage. Therefore, GP behaves like a GA searching for a partial solution (i.e. a good upper part) in a relatively small search space. This means that the algorithm quickly converges towards a common upper part, which cannot later be modified unless mutation is present. At that point the search concentrates on slightly lower levels in the tree with a similar behaviour, until level after level the entire population has completely converged. So, one-point crossover transforms a large search in the original space containing programs with different sizes and shapes into a sequence of smaller quick searches in spaces containing structures of fixed size and shape. An important consequence of the convergence property of GP with one-point crossover is that, like in GAs, mutation becomes a very important operator to prevent premature convergence and to maintain diversity.

Secondly, one-point crossover does not increase the depth of the offspring beyond that of their parents, and therefore beyond the maximum depth of the initial random population. Similarly, one-point crossover will not produce offspring whose depth is smaller than that of the shallowest branch in their parents and therefore than the smallest of the individuals in the initial population. This means that the search performed by GP with one-point crossover and point mutation is limited to a subspace of programs defined by the initial population. Therefore, the initialisation method and parameters chosen for the creation of the initial population can modify significantly the behaviour of the algorithm. For example, if all functions in $\mathcal{F}$ have the same arity and one uses the "full" initialisation method [10] which produces balanced trees with a fixed depth, then the search will be limited to programs with a fixed size and shape. If on the contrary the "ramped half-and-half" initialisation method is used [10], which produces trees of variable shape and size with depths ranging from 0 to the prefixed maximum initial tree depth D, then the entire space of programs with maximum depth D will be searched (at least if the population is big enough).[5]

Thirdly, the offspring produced by one-point crossover inherit the common structure (emphasised with thick lines in Figure 6(c)) of the upper parts of their parents. So, crossing over a program with itself produces the original program. This property is an instance of a more general property: *GP schemata are closed under crossover* (i.e. when one-point crossover is applied to two programs sampling the same schema the offspring produced will sample the same schema, too). This property, which is also known as *respect* [24], is considered

[5]The "ramped half-and-half" initialisation method used in [10] enforced a minimum depth of 2. In our work we use a minimum depth of 0.

to be very important in GAs. A respectful recombination operator makes sure the offspring inherit common characteristics present in the parents. This has the effect of reducing the size of the search space explored by a GA over time. In the presence of selection, this gives a GA the ability to focus the search towards regions of the search space which have shown a consistently higher fitness. Focusing the search quickly is vital if one wants to explore an exponentially large search space in polynomial time.

Fourthly, and perhaps most importantly, one-point crossover makes the calculations necessary to model the disruption of GP schemata feasible. This means that it is possible to study in detail its effects on different kinds of schemata and to obtain a schema theorem as described in the next section.

One-point crossover has some similarity to strong context preserving crossover operator [4]. However, one-point crossover constraints crossover points more tightly than strong context preserving crossover.

Other new operators, like two-point crossover and uniform crossover [22], can easily be defined by extending the basic mechanisms of one-point crossover.

5 GP Schema Theorem

Our schema theorem describes how the number of programs contained in a schema varies from one generation to the next as a result of fitness proportionate selection, one-point crossover and point mutation.

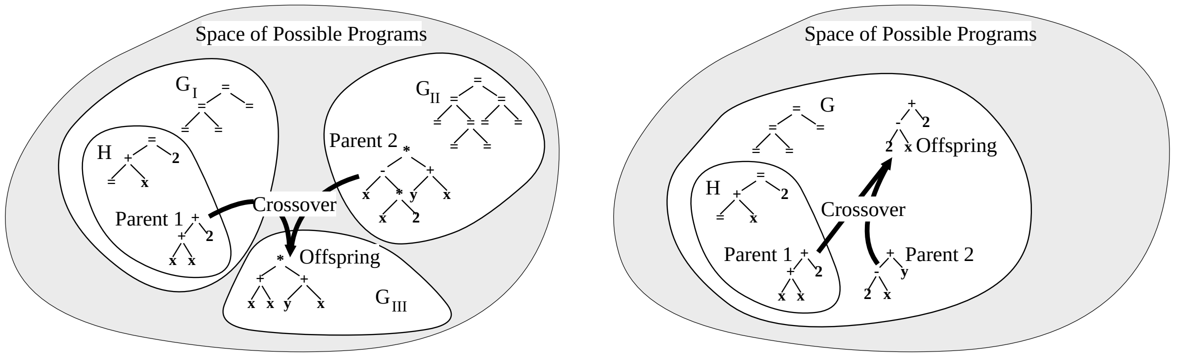
5.1 Effect of Fitness Proportionate Selection

The effect of fitness proportionate selection on our schemata can be obtained by performing exactly the same calculations as for the GA schema theorem (see for example [6, 5, 31] or [29, Appendix C]). If we assume that the mating pool has been created using the roulette-wheel algorithm and we denote with $\{h \in H\}$ the event "a program h sampling the schema H is selected to form the mating pool" then:

$$\Pr\{h \in H\} = \frac{m(H,t)f(H,t)}{M\bar{f}(t)}, \tag{5}$$

where all the symbols have the same meaning as in Equation 1.

As usual, if the population could include a large number of individuals and the ratio between schema fitness and population fitness was constant, this result would support the claim that schemata with above-average (below-average) fitness will tend to get an exponentially increasing (decreasing) number of programs sampling them. However, this claim is often unsupported as, in real-life GAs and GP, populations are



(a) (b)

Figure 7: Two ways in which one-point crossover can disrupt the instances of a schema.

always finite and the estimate of the fitness of schemata changes due to the fact that the composition of the population changes from generation to generation.

5.2 Effect of One-point Crossover

One-point crossover can affect the propagation of schemata by disrupting some of them or creating some new schemata. Let us term $D_c(H)$ the event "H is disrupted when a program h sampling H is crossed over with a program $\hat{h}$" (by "disrupted" we mean that the offspring produced does not sample H).

There are two ways in which H can be disrupted. Firstly, H can be disrupted when h is crossed over with a program $\hat{h}$ with a different structure. If $G(H)$ is the hyperspace associated to H, this can be expressed as the joint event $D_{c1}(H) = \{D_c(H), \hat{h} \notin G(H)\}$. This situation is depicted in Figure 7(a). For example, if we consider the schema H=`(+ = =)` contained in h=`(+ x y)` and we cross h over with $\hat{h}$=`(+ z (+ 2 3))` then swapping the subtree `(+ 2 3)` with `y` would produce the program `(+ x (+ 2 3))` which does not sample $G(H)$=`(= = =)` and therefore cannot sample H.

The probability of the event D_{c1} is given by:

$$\Pr\{D_{c1}(H)\} = \Pr\{D_c(H)|\hat{h} \notin G(H)\} \Pr\{\hat{h} \notin G(H)\}$$

where, by applying Equation 5 (fitness proportionate reproduction) to the schema $G(H)$,

$$\Pr\{\hat{h} \notin G(H)\} = 1 - \Pr\{\hat{h} \in G(H)\} = 1 - \frac{m(G(H),t) f(G(H),t)}{M\bar{f}(t)}.$$

The term $\Pr\{D_c(H)|\hat{h} \notin G(H)\}$, which we will denote with $p_{\text{diff}}(t)$ for brevity, is harder to quantify as not

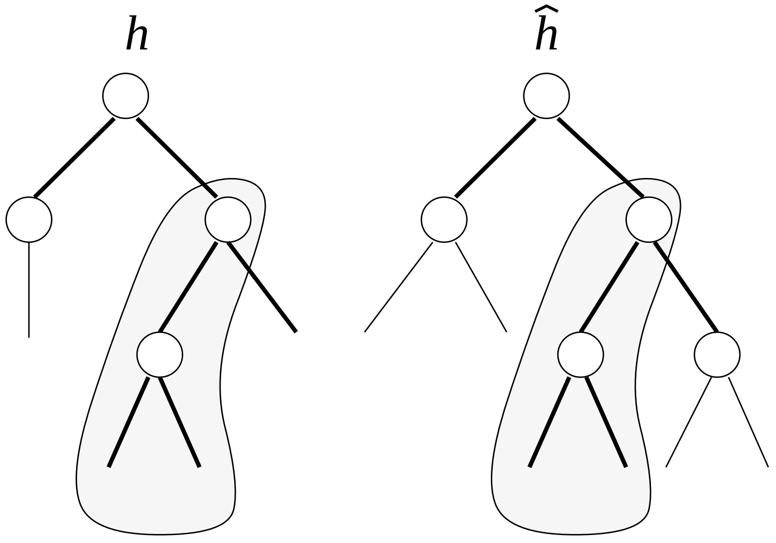


Figure 8: Crossover between two individuals, h and $\hat{h}$, with different shapes. The links where the common crossover point can be placed are drawn with thick lines. The shaded areas enclose the links that, if selected as crossover points, may not lead to the disruption of the schemata in h.

all crossovers between two parents h and $\hat{h}$ with different structure produce offspring which do not sample H. In fact if the common part between h and $\hat{h}$ where the crossover point can be placed includes terminals then crossover can swap subtrees with exactly the same shape and size. This situation is depicted in Figure 8, where crossover points placed in the shaded areas produce subtrees with identical shapes. Swapping such subtrees will produce an offspring which samples $G(H)$. Whether this will sample also H depends on the actual node-composition of the subtrees swapped and on the characteristics of the corresponding subtree in H. For this reason we will leave the term $p_{\text{diff}}(t)$ in symbolic form to avoid using the tempting, but over-simplistic assumption that, when $\hat{h} \notin G(H)$, all crossover operations produce offspring not sampling H, i.e. that $p_{\text{diff}}(t) = 1$. $p_{\text{diff}}(t)$ can be interpreted as the *fragility of the shape* of a schema with respect to the shapes of the other schemata in the population.

The other way in which H can be disrupted is when h is crossed over with a program $\hat{h}$ with the same structure as H (i.e. $\hat{h} \in G(H)$) but which does not sample H (i.e. $\hat{h} \notin H$).[6] This can be expressed as the joint event $D_{c2}(H) = \{D_c(H), \hat{h} \notin H, \hat{h} \in G(H)\}$. This situation is depicted in Figure 7(b).

[6]Given that our schemata are closed under one-point crossover, it is impossible that a schema H be disrupted if $h \in H$ and $\hat{h} \in H$.

The probability of $D_{c2}(H)$ is given by:

$$\Pr\{D_{c2}(H)\} = \Pr\{D_c(H)|\hat{h} \notin H, \hat{h} \in G(H)\} \Pr\{\hat{h} \notin H, \hat{h} \in G(H)\}.$$

Given that the events $\{\hat{h} \in H\}$ are a subset of the events $\{\hat{h} \in G(H)\}$ as $H \subset G(H)$ we can write

$$\Pr\{\hat{h} \notin H, \hat{h} \in G(H)\} = \Pr\{\hat{h} \in G(H)\} - \Pr\{\hat{h} \in H\},$$

where, by applying Equation 5,

$$\Pr\{\hat{h} \in G(H)\} = \frac{m(G(H),t)f(G(H),t)}{M\bar{f}(t)}$$

and

$$\Pr\{\hat{h} \in H\} = \frac{m(H,t)f(H,t)}{M\bar{f}(t)}.$$

A necessary condition for the event $\{D_c(H)|\hat{h} \notin H, \hat{h} \in G(H)\}$ is that the crossover point is between the defining nodes (the non-= nodes) of H: an event that we will term $B(H)$. For example, Table 1 shows which schemata of h=`(+ x y)` could be disrupted depending on which of the two possible crossover points in h is chosen. The probability of $B(H)$ is

$$\Pr\{B(H)\} = \frac{\mathcal{L}(H)}{N(H)-1}.$$

This probability can be interpreted as the intrinsic *fragility of the node composition* of a schema.

Given the previous equations and the inequality

$$\Pr\{D_c(H)|\hat{h} \notin H, \hat{h} \in G(H)\} \leq \Pr\{B(H)\}$$

we obtain:

$$\Pr\{D_{c2}(H)\} \leq \frac{\mathcal{L}(H)}{N(H)-1} \frac{m(G(H),t)f(G(H),t) - m(H,t)f(H,t)}{M\bar{f}(t)}.$$

As the events $D_{c1}(H)$ and $D_{c2}(H)$ are mutually exclusive, the probability of disruption of a schema H due to crossover is simply the sum of $\Pr\{D_{c1}(H)\}$ and $\Pr\{D_{c2}(H)\}$. Given the previous results, the following formula provides an upper bound for such a probability:

$$\Pr\{D_c(H)\} \leq p_{\text{diff}}(t)\left(1 - \frac{m(G(H),t)f(G(H),t)}{M\bar{f}(t)}\right) + \frac{\mathcal{L}(H)}{N(H)-1} \frac{m(G(H),t)f(G(H),t) - m(H,t)f(H,t)}{M\bar{f}(t)}.$$

When crossover is applied to each program in the mating pool with a probability $p_c < 1$, the actual probability of disruption of a schema H due to crossover is $p_c \cdot \Pr\{D_c(H)\}$.

Table 1: Possible effects of crossover on the schemata sampled by the program `(+ x y)` as a function of the crossover point selected when the second parent has the same shape (= = =).

		Crossover Point	
Schema	$\Pr\{B(H)\}$	Between `+` and `x`	Between `+` and `y`
`(= = =)`	0	Unaffected	Unaffected
`(= = y)`	0	Unaffected	Unaffected
`(= x =)`	0	Unaffected	Unaffected
`(+ = =)`	0	Unaffected	Unaffected
`(= x y)`	1	**Disrupted?**	**Disrupted?**
`(+ = y)`	$\frac{1}{2}$	Unaffected	**Disrupted?**
`(+ x =)`	$\frac{1}{2}$	**Disrupted?**	Unaffected
`(+ x y)`	1	**Disrupted?**	**Disrupted?**

5.3 Effect of Point Mutation

Point mutation consists of changing the label of a node into a different one. We assume that it is applied to the nodes of the programs created using selection and crossover as described in the previous sections with a probability p_m per node. A schema H will survive mutation only if *all* its $\mathcal{O}(H)$ defining nodes are unaltered. The probability that each node is not altered is $1 - p_m$. So, a schema will be disrupted by mutation with a probability:

$$\Pr\{D_m(H)\} = 1 - (1 - p_m)^{\mathcal{O}(H)}$$

This equation emphasises that higher order schemata are affected by mutation more than lower order ones.

5.4 Theorem

By considering all the various effects discussed above, we can write:

$$E[m(H, t+1)] = M \Pr\{h \in H\}(1 - \Pr\{D_m(H)\})(1 - p_c \Pr\{D_c(H)\}).$$

By substituting the previous results into this expression we obtain a lower bound for the expected number of individuals sampling a schema H at generation $t + 1$ as stated by the following:

Theorem 6 (GP Schema Theorem). *In a generational GP with fitness proportionate selection, one-point crossover and point mutation:*

$$E[m(H,t+1)] \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)} \cdot (1-p_m)^{\mathcal{O}(H)} \cdot \tag{6}$$
$$\left\{1 - p_c\left[p_{\text{diff}}(t)\left(1 - \frac{m(G(H),t)f(G(H),t)}{M\bar{f}(t)}\right) + \frac{\mathcal{L}(H)}{(N(H)-1)}\frac{m(G(H),t)f(G(H),t) - m(H,t)f(H,t)}{M\bar{f}(t)}\right]\right\}$$

6 Discussion

The GP schema theorem given in Equation 6 is considerably more complicated than the corresponding version for GAs in Equation 1. This is due to the fact that in GP the trees undergoing optimisation have variable size and shape. In Equation 6 this is accounted for by the presence of the terms $m(G(H),t)$ and $f(G(H),t)$, which summarise the characteristics of all the programs in the population having the same shape and size as the schema H whose propagation is being studied. Therefore in GP the propagation of a schema H depends not only on the features of the programs sampling it but also on the features of the programs having the same structure as H. However, the term $p_{\text{diff}}(t)$, which represents the fragility of the shape of the schema H in the population, also plays an important role.

In the following we will discuss how these terms change over time in a run and show that GP with one-point crossover tends asymptotically to behave like a GA and that, for t big enough, Equation 6 transforms into Equation 1.[7] In particular we will consider the situation at the early stages and at late stages of a run.

6.1 Early stages of a GP run

If we consider the typical case in which functions with different arity are present in the function set and the initialisation method is "ramped-half-and-half" or "grow" [10], *at the beginning of a run* it is unlikely that two trees h and $\hat{h}$ with *different* shapes undergoing crossover will swap a subtree with exactly the same shape. This is because given the diversity in the initial population only a small number of individuals will have common parts including terminals (refer to Figure 8). Therefore, we can safely assume that $p_{\text{diff}}(t) \approx 1$.

The term $m(G(H),t)f(G(H),t)$ represents the total fitness allocated to the programs with structure $G(H)$ (by definitions it is equal to $\sum_{h \in G(H)} f(h)$, where $f(h)$ is the fitness of individual h). Likewise $M\bar{f}(t)$

[7]Some of the arguments presented in the rest of the section are the result of observing the actual behaviour of runs of GP with one-point crossover (some of which are described in [21]) and also of the experiments reported in [20] in which we have counted the schemata present in small populations and studied their creation, disruption and propagation.

is the total fitness in the population (i.e. $\sum_{h \in Pop(t)} f(h)$). Therefore the diversity of the population at the beginning of a run implies also that $m(G(H),t)f(G(H),t) \ll M\bar{f}(t)$ and that *the probability of schema disruption is very big (i.e. close to 1) at the beginning of a run*, even disregarding the events $D_{c2}(H)$. Therefore, at the beginning of a run crossover heavily counteracts the effect of selection.

Nonetheless schemata whose shape $G(H)$ is of above average fitness (i.e. $f(G(H),t) > \bar{f}(t)$) and is shared by an above average number of programs will tend to survive more frequently even in this early highly disruptive phase of a run, especially if they have short defining-length (and therefore low order) with respect to their total size.

6.2 Late stages of a GP run

If the mutation rate is small after a while the population in GP with one-point crossover will start converging, like a GA, and the diversity of shapes and sizes will decrease. This in turn will make the common part between pairs of programs undergoing crossover grow and include more and more terminals. As a result, the probability of swapping subtrees with the same structure when crossing over programs with different shape will increase and the fragility of the shape of the schema $p_{\mathrm{diff}}(t)$ will decrease. Losing diversity also means that a relatively small number of different program structures $G(H)$ will survive in the population and that the factor multiplying $p_{\mathrm{diff}}(t)$ in Equation 6 will become smaller.

Therefore, in a second phase of a run the probability of disruption when crossover is performed between trees with the same shape and size, $\Pr\{D_{c2}(H)\}$, will start playing an important role in determining which schemata will propagate according to their fitness and which will instead be disrupted. In particular, a schema H with above average fitness and short defining-length (and therefore low order) with respect to its total size will tend to survive better. However, also schemata with large defining lengths can survive *if they suffer from little competition from the schemata sampled by the programs of the same shape not containing H*. This happens when $m(G(H),t)f(G(H),t) - m(H,t)f(H,t) = \sum_{h \in G(H), h \notin H} f(h) \ll M\bar{f}(t)$.

After many generations these effects are exacerbated: the probability of swapping subtrees with the same structure tends to 1 and $p_{\mathrm{diff}}(t)$ tends to become unimportant. In this situation $m(G(H),t) = M$, $f(G(H),t) = \bar{f}(t)$ and Equation 6 becomes exactly the same as Equation 1. This is consistent with the intuition that if all the programs have the same structure then their nodes can be described with fixed-length strings and GP with one-point crossover is really nothing more than a GA.[8] This asymptotic behaviour of

[8]In less typical conditions where the "full" initialisation method is used and all functions have the same arity, then every

GP with one-point crossover can be summarised with the slogan:

$$\lim_{t\to\infty} \mathrm{GP}_{\mathrm{1pt}}(t) = \mathrm{GA}.$$

6.3 Interpretation

The presence of two parts in the term multiplying p_c in Equation 6 accounts for the fact that in GP with one-point crossover two competitions are going on. The first one happens at the beginning of a run when different *hyperspaces* $G(H)$ representing all programs with a given shape and size compete. In this phase the defining length $\mathcal{L}(H)$ and the number of nodes $N(H)$ of the individual *hyperplanes* H sampling a hyperspace $G(H)$ are nearly irrelevant: only the number of programs in $G(H)$ and their fitness count. The second competition starts when the first one starts settling. In this second phase the hyperplanes within the few remaining hyperspaces start competing on the basis of their relative defining lengths as well as their fitness. In this phase GP tends to behave more and more like a standard GA.

As discussed in [19], in general no schema theorem alone can support conclusions on the correctness of the building block hypothesis. However, our definition of schema and the characterisation of $\Pr\{D_c(H)\}$ presented above seem to suggest that the building block hypothesis is much less troublesome in GP with one-point crossover than in standard GP. It is possible, however, that two kinds of building blocks are needed to explain how GP builds solutions: building blocks representing hyperspaces and building blocks representing hyperplanes within hyperspaces.

7 Conclusions

In this paper we have presented a simplified form of GP in which, thanks to constraints on the selection of crossover points, crossover transmits to the offspring many of the features common to the parents. This property, which is known as respect, may make the search easier by making the fitness lanscape smoother thanks to a higher correlation between the fitness of parents and offspring. It also encourages convergence.

Rather than using the classical strategy of showing experimentally that this form of GP, on a few carefully selected problems, is better than standard GP, we start from developing a schema theory for it. As a natural follow-up of this theoretical work we have recently started an investigation of the performance of GP with one-point crossover obtaining very promising results [21] on the even parity problems. In very recent work

individual in the population has the same shape and Equation 6 is the same as Equation 1 from the beginning of a run.

we have also studied both theoretically and experimentally the search properties of one-point crossover and we have compared them to those of standard crossover (and uniform crossover) [22]. In that work we have shown that one-point crossover may explore the search space of programs better than standard crossover. More research will be needed to see which operator is better on different classes of problems and which operator allows more powerful analyses.

Our schema theory is based on a new simpler definition of the concept of schema for GP which is much closer to the original concept of schema in GAs than the definitions used by other GP theorists. The simplicity of this definition along with that of one-point crossover has allowed us to derive a new schema theorem in which, for the first time, the competitions between programs with different structure and programs with the same structure have been modelled mathematically. These models have been then discussed showing that in the runs of GP with one-point crossover and point-mutation two phases are present: one in which the algorithm searches for the best shape and size for the solutions, and one in which the optimal composition in terms of functions and terminals is sought for. These two phases may overlap and the population moves from one to the other smoothly.

Our analysis shows that the new schema theorem is the natural counterpart for GP of the GA schema theorem and that the GP schema theorem asymptotically tends to it. It also suggests that the building block hypothesis, which has been strongly criticised in the context of standard GP, might in fact hold in GP with one-point crossover.

Some recent experimental results have corroborated the theory presented in this paper [20]. More theoretical and experimental work will be needed to investigate the building block hypothesis.

Acknowledgements

The authors wish to thank Una-May O'Reilly and the members of the EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group for useful discussions and comments. The reviewers are also thanked for their valuable help in improving this paper. This research was partially supported by a grant under the British Council-MURST/CRUI agreement and by a contract with the Defence Research Agency in Malvern.

References

[1] Lee Altenberg. The Schema Theorem and Price's Theorem. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[2] Peter J. Angeline and K. E. Kinnear, Jr., editors. *Advances in Genetic Programming 2*. MIT Press, Cambridge, MA, USA, 1996.

[3] Kenneth A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975. Dissertation Abstracts International 36(10), 5140B, University Microfilms Number 76-9381.

[4] Patrik D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[5] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[6] John Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts, second edition, 1992.

[7] Hillol Kargupta. Signal-to-noise, crosstalk, and long range problem difficulty in genetic algorithms. In Larry J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 193–200, Pittsburgh, PA, USA, July15–19 1995. Morgan Kaufmann Publishers.

[8] Hillol Kargupta and David E. Goldberg. Decision making in genetic algorithms: A signal-to-noise perspective. Technical Report IlliGAL Report No 94004*, Illinois Genetic Algorithms Lab (?), 1994.

[9] Kenneth E. Kinnear, Jr., editor. *Advances in Genetic Programming*. MIT Press, 1994.

[10] John R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[11] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[12] John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors. *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[13] John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors. *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[14] William B. Langdon. A bibliography for genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter B, pages 507–532. MIT Press, Cambridge, MA, USA, 1996.

[15] William G. Macready and David H. Wolpert. On 2-armed gaussian bandits and optimization. Sante Fe Institute Working Paper 96-05-009, March 1996.

[16] Ben McKay, Mark J. Willis, and Geoffrey W. Barton. Using a tree structured genetic algorithm to perform symbolic regression. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, volume 414, pages 487–492, Sheffield, UK, 12-14 September 1995. IEE.

[17] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[18] Una-May O'Reilly. *An Analysis of Genetic Programming*. PhD thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 September 1995.

[19] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[20] Riccardo Poli and W. B. Langdon. An experimental analysis of schema creation, propagation and disruption in genetic programming. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 18–25, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.

[21] Riccardo Poli and W. B. Langdon. Genetic programming with one-point crossover. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag London, 23-27 June 1997.

[22] Riccardo Poli and William B Langdon. On the search properties of different crossover operators in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann. Forthcoming.

[23] Nicholas J. Radcliffe. Forma analysis and random respectful recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.

[24] Nicholas J. Radcliffe. The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 10:339–384, 1994.

[25] Justinian P. Rosca. Analysis of complexity drift in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286–294, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[26] Nicol N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9(1):9–21, 1992.

[27] Cees H.M. van Kemenade. Cross-competition between building blocks, propagating information to subsequent generations. In Thomas Back, editor, *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 1–8, Michigan State University, East Lansing, MI, USA, 19-23 July 1997. Morgan Kaufmann.

[28] Peter A. Whigham. A schema theorem for context-free grammars. In *1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 178–181, Perth, Australia, 29 November - 1 December 1995. IEEE Press.

[29] Peter A. Whigham. *Grammatical Bias for Evolutionary Learning*. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, 14 October 1996.

[30] Peter A. Whigham. Search bias, language bias, and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[31] Darrell Whitley. A genetic algorithm tutorial. Technical Report CS-93-103, Department of Computer Science, Colorado State University, August 1993.

[32] Darrell Whitley, K. Mathias, and L. Pyeatt. Hyperplane ranking in simple genetic algorithms. In Larry J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 231–238, San Francisco, July15–19 1995. Morgan kaufmann Publishers.